Linear Regression

S. Sumitra

Notations: x_i: ith data point; x^T: transpose of x; x_{ij} : ith data point's jth attribute.

Let $\{(x_1, y_1), (x_2, y_2)...(x_N, y_N)\}$ be the given data, $x_i \in \mathcal{D}$ and $y_i \in \mathcal{Y}$. Here $\mathcal{D}$ denote the space of input values and Y denote the space of output values.

If the target variable is continuous, the learning problem is known as regression. If the target variable y takes only discrete values, it is a classification problem.

In this paper, $\mathcal{D}$ =$\mathbb{R}^n$ and $\mathcal{Y}$= $\mathbb{R}$.

The simplest linear model is the representation of f as a linear combination of x. That is,

$$f(x_i) = w_0 + w_1 x_{i1} + w_2 x_{i2} + \ldots w_n x_{in} \tag{1}$$

where, $x_i = (x_{i1}, x_{i2}, \ldots x_{in})^T$ and $w = (w_0, w_1, \ldots, w_n)^T \in \mathbb{R}^{n+1}$. Here, $w_i's$ are the parameters, parameterizing the space of linear functions mapping from $\mathcal{X}$ to $\mathcal{Y}$.

(1) is the equation of a hyperplane.

By taking $x_{i0} = 1$, (1) can be written as

$$f(x_i) = \sum_{j=0}^{n} w_j x_{ij} = w^T x_i \tag{2}$$

Given a training set, how to choose the values of $w_i's$?

As there are N data points, we could write N equations such that,

$$f(x_i) = \sum_{j=0}^{n} w_j x_{ij} = w^T x_i, i = 1, 2, \ldots N \tag{3}$$

Define the design matrix to be

$$X = \begin{bmatrix} (x_1)^T \\ (x_2)^T \\ . \\ . \\ . \\ (x_N)^T \end{bmatrix}.$$

Here, $x_i's$ are the training data (each x_i is a n dimensional vector). X is a $N \times n+1$ matrix, if we include the intercept term, that is if we set $x_i = (x_{i0}, x_{i1}, ...x_{in})^T$ and set $x_{i0} = 1$. Let y be the N dimensional vector that contains all the target values, that is,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_N \end{bmatrix}.$$

Now the matrix representation for (3) is

$$Xw = y \tag{4}$$

The range space of X is spanned by the columns of X. If X is a square matrix and inverse exists, $w = X^{-1}y$. This might not be the case always. (4) may have no solution or more than one solution. The former case happens when X is not onto and later case happens X is not one to one.

Next we consider the case of rectangular matrices.

1 y is not in the range of X $(n+1 < N)$

We will first consider the case when y is not in the range of X. In this case we find the preimage of the projection of y onto the range space of X. That is, the optimal solution is

$$w^* = \arg \min_{w \in \mathbb{R}^{n+1}} J(w)$$

where

$$J(w) = \frac{1}{2}(d(Xw, y))^2 = \frac{1}{2}||Xw - y||^2 \tag{5}$$

$J(w)$ is called the least square cost function.

Now $J(w) = \frac{1}{2}(d(Xw, y))^2 = \frac{1}{2}||Xw - y||^2 = \frac{1}{2}\langle Xw - y, Xw - y\rangle$

At the minimum value of w, $\nabla J = 0$. That is ,

$$\nabla J \;\; = \;\; X^T Xw - X^T y = 0$$

Hence,

$$X^T X w = X^T y \tag{6}$$

(6) is called the normal equation. Hence,

$$w = (X^T X)^{-1} X^T y \tag{7}$$

provided $(X^T X)^{-1} X^T$ exists. $(X^T X)^{-1} X^T$ is called the pseudo-inverse. For determining w using derivative method, the inverse of $X^T X$ is to be found, which is not computationally effective for large data sets. Hence we resort to iterative search algorithms for finding w.

1.1 Least Mean Squares Algorithm

For finding w that minimizes J, we apply an iterative search algorithm. An iterative search algorithm that minimizes $J(w)$, starts with an initial guess of w and then repeatedly change w to make J(w) smaller, until it converge to the values that minimizes $J(w)$. We consider gradient descent for finding w.

[*Gradient descent: If a real valued function $F(x)$ is defined and differentiable in a neighbourhood of point a, then $F(x)$ decreases fastest if one goes from a in the direction of the negative gradient of F at a, $\nabla F(a)$. The gradient of the function is always perpendicular to the contour lines.(A contour line of a function of two variables is a curve along which the function has a constant value.)*]

(5) can also be written as

$$J(w) = \frac{1}{2} \sum_{i=1}^{N} (f(x_i) - y_i)^2 \tag{8}$$

For applying gradient descent, consider the following steps. Choose an initial $w = (w_0, w_1, ...w_n)^T \in R^{n+1}$. Then repeatedly performs the update

$$w := w - \alpha \nabla J \tag{9}$$

Here, $\alpha > 0$ is called the learning rate.

Since f is a function of $w_0,\ w_1,\ \ldots,\ w_n$, J is a function of $w_0,\ w_1,\ \ldots,\ w_n$. Therefore,

$$\nabla J = \left(\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial w_n} \right)^T \tag{10}$$

Sub: (10) in (9),

$$(w_0, w_1, ...w_n)^T := (w_0, w_1, ...w_n)^T - \alpha \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \ldots, \frac{\partial J}{\partial w_n} \right)^T \tag{11}$$

Hence,

$$w_j := w_j - \alpha \frac{\partial J}{\partial w_j}, j = 0, 1, \ldots n \tag{12}$$

Now

$$\begin{aligned} \frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_{i=1}^{N} (f(x_i) - y_i)^2 \\ &= \sum_{i=1}^{N} (f(x_i) - y_i) \frac{\partial}{\partial w_j} \left(\sum_{j=0}^{n} w_j x_{ij} \right) \\ &= \sum_{i=1}^{N} (f(x_i) - y_i) x_{ij} \end{aligned}$$

Therefore,

$$w_j := w_j + \alpha \sum_{i=1}^{N} (y_i - f(x_i)) x_{ij}, j = 0, 1, 2, \ldots n \tag{13}$$

(13) is called LMS (least mean squares) update or Widrow -Hoff learning rule. Pseudocode of the algorithm can be written as:

Iterate until convergence {

$$w_j := w_j + \alpha \sum_{i=1}^{N} (y_i - f(x_i)) x_{ij}, j = 0, 1, 2, \ldots n$$

}

The magnitude of the update of parameter is proportional to the error term $(y_i - f(x_i))$. Larger change to the parameter is made when the error is large and vice versa.

For updating the parameter, the algorithm looks at every data point in the training set at every step and hence it called batch gradient descent. In general, gradient descent does not guarantee a global minimum. However as J is a convex quadratic function, the algorithm converges to the global minimum (assuming α is not too large).

There is an alternative to batch gradient descent called stochastic gradient descent which can be stated as follows:

Iterate until convergence {

for i=1 to N{

$$w_j := w_j + \alpha(y_i - f(x_i))x_{ij}, j = 0, 1, 2, \ldots n$$

}}

In contrast to batch gradient, stochastic gradient process only one training point at each step. Hence when N becomes large, that is, for large data sets, stochastic gradient descent is more computationally efficient than batch gradient descent.

2 y has more than one pre-image ($N < (n+1)$)

The second case is when y is in the range of X but has more than one pre-image. As there would be more than one w that satisfies the given equation, the following constrained optimization problem has to be considered:

$$\begin{aligned} \underset{w \in \mathbb{R}^{n+1}}{\text{minimize}} \quad & ||w||^2 \\ \text{subject to} \quad & Xw = y \end{aligned} \tag{14}$$

By applying lagrangian theory,

$$L(w, \lambda) = ||w||^2 + \lambda^T(Xw - y)$$

where $\lambda^T = (\lambda_1, \lambda_2, \ldots \lambda_N)$. $\lambda_i, i = 1, 2, \ldots N$ are the lagrangian parameters. By equating $\dfrac{\partial L}{\partial w} = 0$

$$2w + X^T\lambda = 0$$

Hence

$$w = -\frac{X^T\lambda}{2} \tag{15}$$

By equating $\dfrac{\partial L}{\partial \lambda} = 0$ we get,

$$Xw - y = 0 \tag{16}$$

Using (15), the above equation becomes

$$\frac{-XX^T\lambda}{2} = y$$

Therefore

$$\lambda = -(XX^T)^{-1}2y \tag{17}$$

Sub: (17) to (15),

$$w = X^T(XX^T)^{-1}y$$

References

(1)*Andrew Ng's lecture note (CS 229).*
(2) Bishop's Book: Pattern Recognition and Machine Learning.